AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN

Gavrilut, Voica; Zhao, Luxi; Raagaard, Michael L.; Pop, Paul

Published in:
IEEE Access

Link to article, DOI:
10.1109/ACCESS.2018.2883644

Publication date:
2018

Document Version
Peer reviewed version

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 5, 2018, accepted October 27, 2018, date of publication November 27, 2018,
date of current version December 27, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2883644

AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN

VOICA GAVRILUŢ, LUXI ZHAO, MICHAEL L. RAAGAARD, AND PAUL POP
DTU Compute, Technical University of Denmark, 2800 Kongens Lyngby, Denmark

Corresponding author: Voica Gavriluţ (voga@dtu.dk)

This work was supported in part by the ENABLE-S3 Project from the ECSEL Joint Undertaking under Grant 692455.

ABSTRACT IEEE 802.1 time-sensitive networking (TSN) is a set of amendments to the IEEE 802.1 standard that enable safety-critical and real-time behavior over Ethernet for the industrial automation and automotive domains. Selected TSN mechanisms offer the possibility to emulate the well-known traffic classes found in mixed-criticality distributed systems: Time-triggered (TT) communication with low jitter and bounded end-to-end latency, audio-video-bridging (AVB) streams with bounded end-to-end latency, and general best-effort messages, which have no timing guarantees. Critical traffic is guaranteed via the global network schedule which is stored in so-called gate control lists (GCLs) and controls the timely behavior of frames for each queue of an egress port. Although researchers have started to propose approaches for the routing and scheduling (i.e., GCL synthesis) of TT traffic, all previous research has ignored lower priority real-time traffic, such as AVB, resulting in TT configurations that may increase the worst-case delays of AVB traffic, rendering it unschedulable. In this paper, we propose a joint routing and scheduling approach for TT traffic, which takes into account the AVB traffic, such that both TT and the AVB traffic are schedulable. We extensively evaluate our approach on a number of synthetic and realistic test cases.

INDEX TERMS IEEE 802.1 Time-sensitive networking, deterministic ethernet, real-time networks, routing, scheduling, meta-heuristic optimization.

I. INTRODUCTION

In this paper, we are interested in safety-critical and real-time applications implemented using distributed cyber-physical systems. Several real-time capable communication protocols have been proposed and are in use in different application areas, e.g., FlexRay for automotive [8], ARINC 664 p7 for avionics [2], and EtherCAT for industrial automation [16]. However, emerging applications, e.g., Advanced Driver Assistance Systems (ADAS), autonomous driving, or industrial automation, have increasing bandwidth and real-time demands. For instance, autonomous driving requires data rates of at least 100 Mbps for graphical computing based on camera, radar, and Light Detection And Ranging (LIDAR) data, whereas CAN and FlexRay only provide data rates of up to 1 Mbps and 10 Mbps, respectively.

The well-known networking standard *IEEE 802.3 Ethernet* [14] meets the emerging bandwidth requirements for a wide range of application areas, while remaining scalable and cost-effective. It does, however, lack real-time and dependability capabilities [6]. Many extensions, such as EtherCAT, PROFINET, ARINC 664p7 [3], and TTEthernet [24], have been suggested and are used in the industry. Although they satisfy the timing requirements, they are incompatible; hence, the interoperability within the same network is not possible without losing real-time guarantees [7]. To mitigate this drawback, the *IEEE 802.1 Time-Sensitive Networking (TSN)* Task Group [15] has been working on defining standard amendments for real-time and safety-critical enhancements over Ethernet.

In standard switched Ethernet networks, in which end-systems are interconnected through a series of physical links and bridges (switches), communication from one sender to one or multiple receivers (flows/streams) is done via frames that are forwarded via a route through the network. Standard switching fabric contains queues on the egress ports of the switches (and end-systems) which implement a standard priority scheme and which store frames until the port is free for transmission. Hence, a frame might experience queueing delay while waiting for the transmission of higher priority frames and earlier arriving frames with same priority. This leads to network congestion causing nondeterministic behavior and variance in frame arrival times.

TSN: First, *IEEE 802.1Q-2005*[1] introduced support for prioritizing the Best-Effort (BE) traffic providing some higher Quality-of-Service (QoS) properties. The *IEEE Audio-Video-Bridging (AVB)* Task Group [13] has developed another enhancement generally known as AVB which introduce two new shaped AVB traffic classes enabling bounded latency. In 2012, *TSN Task Group* has started extending the protocol towards safety-critical and time-sensitive applications which require even more stringent real-time guarantees. Via selected amendments defined in TSN, i.e., *IEEE 802.1Qbv Enhancements for Scheduled Traffic* and *IEEE 802.11ASrev Clock synchronization*, the well-known Time-Triggered (TT) traffic class, which has strict jitter and end-to-end latency guarantees, can be emulated in standardized TSN networks.

TT Traffic: requires schedule tables, called *Gate Control Lists* (GCLs), that defines the exact queue transmission times of frames on every egress port along the route of the respective flows. The schedules define at which points in time a so-called timed-gate that is associated with every queue is opened and when it is closed, enabling and disabling frame transmission, respectively. Since the schedules in different devices need to be aligned, a clock synchronization mechanism is required in order to provide a global time reference. This synchronization protocol is defined in the *IEEE 802.1ASrev* standard and, together with *IEEE 802.1Qbv*, provide the basic building blocks for achieving determinism and bounded end-to-end latency for critical traffic. **AVB traffic** is intended for applications that require bounded end-to-end latencies, but that do not have the same stringent real-time requirements as TT traffic. In order to prevent the starvation of lower priority messages AVB introduces two new shaped traffic classes (AVB Class A and B) and uses the Credit-Based Shaper (CBS) defined in *IEEE 802.1BA*. The worst-case end-to-end delays (WCDs) of AVB flows can be analyzed via timing analysis methods, e.g., based on Network Calculus. The AVB traffic type is especially useful for industrial applications, which require dynamic reconfiguration to meet new business demands, allowing computation and communication services to evolve over time with minimal disruption. Finally, **BE traffic** has the lowest priority and does not provide any timing guarantees.

Note that for implementing real-time applications, both TT and AVB traffic types provide bounded latencies, and the choice of traffic type for a message depends on the particularities of the application. The WCDs of TT flows are determined by the GCLs. However, the WCD of an AVB message depends on the GCLs of TT traffic and the other AVB messages that share the same queue or have a higher priority. Recent timing analysis work for AVB [31] has shown how to determine the WCDs of AVB messages taking into account the impact of TT traffic via the GCLs.

Problem Formulation: In this paper we consider real-time applications implemented using both the TT and AVB traffic types running on TSN-based distributed architectures. We assume that the network topology is given. The applications messages are modelled as TT and AVB messages. Furthermore, we assume that the traffic type of each message is given. We are interested to synthesize the routing and the GCLs for TT traffic, such that both TT and AVB traffic is schedulable.

[1]We will not provide references for all sub-standards, but these can be easily found based on their names via IEEE Xplore.

A. RELATED WORK AND CONTRIBUTION

There is a lot of research work on deriving schedule tables for tasks and messages [4]. For TTEthernet, researchers have proposed strategies for the scheduling of TT frames on network links [26], which take into account the lower priority Rate-Constrained (RC) traffic type of TTEthernet. However, although there are similarities between TSN and TTEthernet, they differ in some significant aspects: Messages in TTEthernet consist of a single frame, whereas TSN messages may consist of multiple frames. Furthermore, TTEthernet schedule tables are specified for individual TT frames, whereas TSN specifies schedules for the output port queues, not frames. Consequently, all frames sharing the same queue are affected by the associated GCL. As a result, the work on TTEthernet scheduling is not directly applicable to TSN. In addition, the transmission of RC frames and the corresponding timing analysis for WCDs differ significantly compared to AVB.

For the GCL synthesis problem in TSN, researchers have started to propose several approaches, based on, e.g., Satisfiability/Optimization Modulo Theories (SMT/OMT) [5] and metaheuristics [7]. Deciding the routing of traffic flows is also an important problem. The TSN standards proposes dynamic routing and reservation mechanisms, such as *IEEE 802.1Qca* and *IEEE 802.1Qcc*. Such dynamic routing is appropriate for non-critical traffic. However, real-time and safety-critical traffic uses static routes, decided at design time. Hence, researchers have also addressed the problem of determining the static routes for both TT [18] and AVB [17] traffic. Researchers have addressed also the joint routing and scheduling problem, proposing solutions based on Integer Linear Programming (ILP) [25] and a List Scheduling-based heuristic [20].

However, all of the approaches for routing or GCL synthesis of fully deterministic TT transmission in TSN presented previously have looked at TT traffic in isolation, completely ignoring the impact on AVB traffic. The work in [12] so far is the only one which addresses the GCL synthesis for mixed-criticality applications in TSN. As we will show in the experimental results section, ignoring AVB traffic results in routes and GCLs that are optimized for TT at the expense of AVB traffic, which leads to very large WCDs for AVB.

Contribution: In this paper, we propose a joint routing and scheduling of TT traffic in TSN taking into account the AVB traffic. To the best of our knowledge this is the first work dealing with the interdependence between AVB traffic and the TT routing and scheduling. Considering the effect of

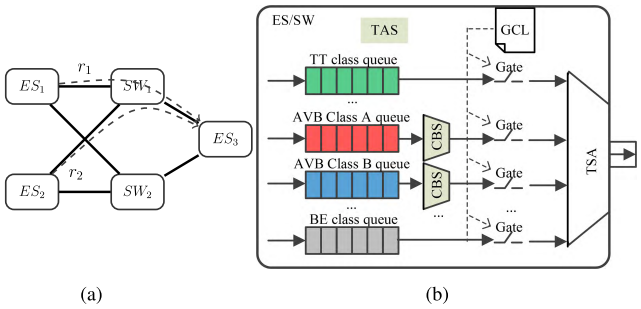


FIGURE 1. TSN network and device architecture. (a) Example architecture. (b) A TAS for an output port [31].

TT routing and scheduling on AVB traffic results optimized solutions guarantee the schedulability of TT traffic while at the same time reducing the WCDs of AVB traffic, such that its end-to-end latency requirements are fulfilled. To solve this problem we have developed a solution that integrates a K-Shortest Path (KSP) heuristic for routing with a Greedy Randomized Adaptive Search Procedure-based (GRASP)-based metaheuristic for scheduling.

The paper is structured as follows: Section II presents the architecture and applications models. We introduce briefly TSN and present how TT and AVB work in section III. Section IV outlines our problem formulation and section V presents our proposed solution. The experimental results are in section VI and the last section presents our conclusions.

II. SYSTEM MODELS

A. ARCHITECTURE MODEL

The architecture model is an abstract representation of the physical TSN network, including end systems, switches, and physical links. The topology is modeled as an undirected graph G, where the vertices represent devices in the network, i.e., end systems $\mathcal{ES}$, and network switches $\mathcal{SW}$, known in TSN also as bridges. The edges represent physical full-duplex links. A data link $dl_{i,j}$ is a directed communication link from a vertex v_i to another vertex v_j. Fig. 1a shows the topology of a network with three end systems, ES_1, ES_2, ES_3, and two switches, SW_1 and SW_2. A route r_i is a cycle-free ordered sequence of data links connecting one sending end system with one or more receiving end systems, via switches. Without loss of generality, we consider in this paper that the routes are unicast. Fig. 1a shows two routes: $r_1 = \{ES_1, SW_1, ES_3\}$, and $r_2 = \{ES_2, SW_1, ES_3\}$. The set of all routes in a network is denoted $\mathcal{R}$.

B. APPLICATION MODEL

The real-time applications are modeled as a set of messages which can be transmitted as TT or AVB flows. The set of flows in the system is denoted as $\mathcal{F} = \mathcal{F}^{TT} \cup \mathcal{F}^{AVB}$. Associated with each flow f_i is the tuple of attributes (v_s, v_t, T, D, P), where v_s denotes the sending end system and v_t denotes the receiving end system. The flows are periodic, with a period T and have a relative deadline D. P is the payload, or data size, of f_i. A single Ethernet frame transmits a payload of at most 1500 bytes (B), the so-called Maximum Transmission Unit (MTU). If the data size is larger than MTU, the message is fragmented into multiple frames, f_i^k denoting the k^{th} frame of the flow f_i. Table 1 shows eight sample flows, four TT flows f_1 to f_4 and four AVB flows f_5 to f_8.

TABLE 1. Example application model.

flow	type	v_s	v_t	T (μs)	D (μs)	P (B)
$f_1 \cdots f_4$	TT	ES_1	ES_3	150	150	750
$f_5 \cdots f_8$	AVB	ES_2	ES_3	150	100	1500

A routing $R : \mathcal{F} \mapsto \mathcal{R} \cup \{\emptyset\}$ is a function which maps a flow to the route on which that message is forwarded. To show that a flow f_i has no assigned route we use the notation $R(f_i) = \emptyset$. In this work the routing R has to be decided. $U(R, dl_{i,j})$ denotes the utilization on link $dl_{i,j}$ for the routing R. It represents the sum of the bandwidth of the flows routed through the link $dl_{i,j}$ and is defined as:

$$U(R, dl_{i,j}) = \sum_{f_k \in \mathcal{F} | dl_{i,j} \in R(f_k)} \frac{f_k.P}{f_k.T}.$$

For a route $r \in R$ the utilization U represents the maximum utilization of links composing the route, i.e., $U(R, r) = \max_{dl_i \in r} U(R, dl_i)$.

III. TSN PROTOCOL

TSN is based on the switched multi-hop network architecture from *IEEE 802.3 Ethernet*. Switches interconnect end systems via *full-duplex* links, meaning that the physical links enable transmission in both directions simultaneously.

Ethernet frames contain *IEEE 802.1Q* headers, with two fields of importance to TT traffic:

- VLAN Identifier (VID) is a 12-bit field specifying the Virtual LAN of a frame. This is used to distinguish frames from different messages.
- Priority Code Point (PCP) is a 3-bit field specifying the priority level, i.e., the traffic class such as TT, AVB, or BE. Furthermore, it defines which queue the frame is assigned to within a switch.

An Ethernet switch has ingress (incoming) and egress (outgoing) ports connecting it via links to surrounding switches and end systems. Each egress port typically has eight queues for storing frames that wait to be forwarded on the corresponding link, one or more TT queues, two for AVB (Class A and B respectively) and the remaining queues are used for BE. Fig. 2 shows part of a TSN network.

A. TT TRAFFIC

IEEE 802.1ASrev provides a clock synchronization protocol to obtain a global time base for TT transmission. Taking advantage of the global synchronized clock, *IEEE 802.1Qbv* defines a Time-Aware Shaper (TAS) to achieve low latency for TT traffic by establishing completely independent time windows by opening and closing the gates. Interference from

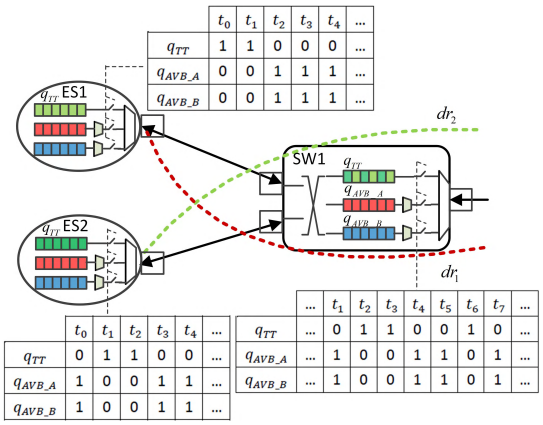


FIGURE 2. TSN network with internal queues, gates and GCLs [31].

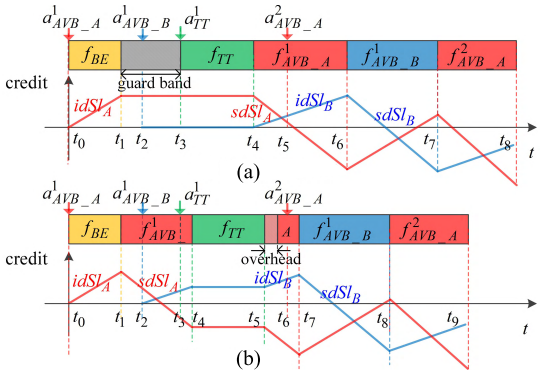


FIGURE 3. Example AVB transmission [31].

lower priority traffic is prevented by closing the gates of the remaining queues, see Fig. 1b. When the egress port is idle, the next frame is selected for transmission from the queue with highest priority among the queues with open gates. Opening queues in a mutually-exclusive fashion, allows for full control of forwarded frame.

A Gate Control List (GCL) defines for each egress port, when the queue gates are open and closed. In Fig. 2 they are depicted as tables. 1 and 0 in the GCL represent an open and closed gate, respectively. Using the GCLs to schedule forwarding of frames in a route from sender to receiver, enables very low latency and jitter for TT traffic, making it suitable for hard real-time communication.

The GCLs can be constructed in such a way that AVB and BE traffic are prevented from initiating transmission in time slots reserved for TT frames. However, nondeterminism could still occur due to interference with other TT flows. When a frame is scheduled for transmission on a link in a given time interval, the corresponding GCL is set to open the associated gate in that interval. Suppose something goes wrong, so the frame is not fully received, or is not the first frame in the queue as expected. Then the link transmits the wrong frame or remains idle when it should be transmitting. Consequently, nondeterminism is introduced, which means timeliness is compromised, see [5] for an in-depth discussion. Similar to the related work on GCL synthesis [5], we will determine the GCLs such that the non-determinism is avoided, see section IV-A for a discussion.

Integration Modes: When there are mixed-criticality frames within the same network, TT traffic might be delayed by AVB or BE traffic that is already being transmitted at the time of the schedule trigger for TT (i.e., gate open for the respective queue). In order to reduce this delay, TSN introduces two mechanisms. The first is referred to as *non-preemption* mode. The gate of lower-priority traffic can be closed in advance of the TT schedule event such that the port is available for the TT traffic. This mechanism is similar to the "guard band" approach found in TTEthernet [4]. The second mechanism is *preemption*, defined by *IEEE 802.1Qbu*, where the transmission of an AVB (or BE) frame will be interrupted by the transmission of a TT frame and resumed once the TT frame has been fully transmitted. As specified in *IEEE 802.1Qbu* even in the case of preemption mode the fragments of the preempted lower priority frames should be transmitted by well formatted Ethernet frames, i.e., the so-called smallest non-preemptable fragment consists of an Ethernet preamble, a minimum of 64 B of data and the trail which further consists on a CRC code and an inter frame gap [27]. Please note that for the first integration mode the delay of TT frames is 0 while for the second mode it is upper bounded by the transmission duration of the smallest non-preemptible fragment.

B. AVB TRAFFIC

The availability of an AVB queue is also determined by a Credit-Based Shaper (CBS) and the purpose of CBS is to prevent the starvation of lower priority flows. Hence, an enqueued AVB frame is allowed to be transmitted if (i) the queue gate is open, (ii) the CBS allows it and (iii) there are no other higher priority AVB frames being transmitted.

The CBS standardized in *IEEE 802.1Qat* in conjunction with the amendments in *IEEE 802.1Qbv* makes the queue available for transmission whenever the amount of *credit* is positive or zero. The credit is initially zero, it is decreased with a *sending slope* (*sdSl*) while transmitting and frozen while the gate is closed. Transmission is only initiated when credit is non-negative. The credit is increased with an *idle slope* (*idSl*) when frames are waiting, but they are not being transmitted. If the queue is emptied while the credit is positive, the credit is reset to zero. The idle and sending slopes are configuration parameters described in *IEEE 802.1Qbv*; the idle slope is defined as fraction of link speed and the sending slope as difference between idle slope and link speed.

Using the example in Fig. 3 we show how CBS works, considering also TT and BE traffic. Rectangles on the first timeline represent the transmission of frames and down arrows on top give the frames arrival times, for example $a^i_{AVB\_A}$ indicates the arrival time of the frame f_i. The lines on the timeline show the variation of credit for respective AVB class, where AVB Class A and B are respectively shown with

red and blue. Fig. 3a considers the non-preemption integration mode. An AVB Class A frame $f^1_{AVB\_A}$ arrives at t_0; meanwhile, a BE frame is transmitting. Due to the non-preemption of BE frames, $f^1_{AVB\_A}$ has to wait until f_{BE} finishes its transmission, and credit A is increased with the idle slope $idSl_A$. At time t_1, the transmission of f_{BE} is completed. However, the AVB gates are closed due to the reservation for TT traffic and insufficient idle interval (caused by the guard band) for the whole frame $f^1_{AVB\_A}$ transmission. Therefore, credit A is frozen during $[t_1, t_4]$ when AVB gates are closed. When an AVB Class B frame $f^1_{AVB\_B}$ arrives at t_2, its credit is also frozen. From time t_4, since the gate for TT queue is closed and due to the higher priority of Class A, $f^1_{AVB\_A}$ is allowed to be transmitted. The credits for A and B are, respectively, decreased and increased with the sending slope $sdSl_A$ and idle slope $idSl_B$. During the transmission of $f^1_{AVB\_A}$, another frame $f^2_{AVB\_A}$ is enqueued in the Class A queue at time t_5. Then, at t_6 when frame $f^1_{AVB\_A}$ finishes, there are two frames $f^2_{AVB\_A}$ and $f^1_{AVB\_B}$ waiting to be transmitted. But credit A at this time is negative, therefore $f^2_{AVB\_A}$ is not allowed to be transmitted and $f^1_{AVB\_B}$ has the permission to be transmitted. At the end of $f^1_{AVB\_B}$ transmission, $f^2_{AVB\_A}$ starts its transmission as credit A has been increased to a non-negative value.

In Fig. 3b, we present the preemption mode. The arrival times are the same as in Fig. 3a. However, due to the preemption integration mode, the TT frame f_{TT} is delayed from t_3 to t_4, and when $f^1_{AVB\_A}$ resumes its transmission after being preempted, we have to consider an additional overhead (depicted).

IV. PROBLEM FORMULATION

The problem addressed in this paper is: Given a TSN network topology G, and a set of TT and AVB flows $\mathcal{F} = \mathcal{F}^{TT} \cup \mathcal{F}^{AVB}$ determine an implementation ϕ such that: (i) all the flows are schedulable, i.e., the $WCD(f_i) \leq f_i.D$ for all TT and AVB flows. Determining ϕ means: (1) deciding the route $R(f_i)$ for each TT flow $f_i \in \mathcal{F}^{TT}$, (2) deciding the number of TT queues, (3) mapping the TT flows to egress port TT queues and (4) deriving the GCLs $\mathcal{GCL}$.

Note that mapping of AVB flows to AVB queues is decided by their class, i.e., AVB Class A flows are assigned to AVB Class A queue and B flows to Class B queue. Our proposed solution can also determine the AVB routes at the same time with the TT routes. However, the focus of this paper is on determining the routing and scheduling of TT flows, so we consider the AVB routing given and fixed. The routing of AVB flows aiming at reducing their WCDs has been addressed by us in [17].

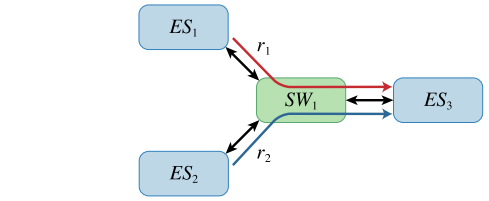
A. GCL SYNTHESIS FOR TT

Let us consider the example from Fig. 4 where we have the two flows from Fig. 4a routed as indicated in Fig. 4b.

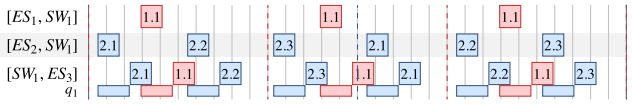
Fig. 4 shows GCLs using a Gantt chart, depicting how TT frames are transmitted. The x-axis represents time dimension, while y-axis is related to output ports of nodes.

flow	v_s	v_t	r	$T(\mu s)$	$D(\mu s)$	$P(B)$
f_1	ES_1	ES_3	r_1	100	100	1500
f_2	ES_2	ES_3	r_2	150	150	4500

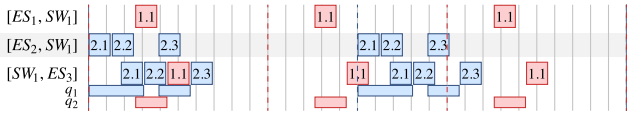
(a)



(b)



(c)



(d)

FIGURE 4. Example GCL synthesis for TT. (a) Set of TT flows with attributes. (b) Network topology and routing of TT flows. (c) Minimum queue usage. (d) Minimum end-to-end latency.

Moreover, the rectangles represent TT frames' transmission. The left side of rectangle is the start time of the transmitted frame, and its width represents the transmission duration which is related to the frame size and the physical link rate. To illustrate the queue usage, we use thin rows labeled q_i below the link schedules showing when frames are in the queues of the respective egress port.

The GCLs must satisfy the following constraints:

- *Link congestion.* A data link is limited by its hardware to only transmit a single frame at a time, i.e., frames on the same link cannot overlap in the time domain. This corresponds to the property that boxes on the same row of Fig. 4 do not overlap. The link can be seen as a shared resource that can only be occupied by a single frame at a time.
- *Flow transmission.* A switch cannot forward a frame until the entire frame has been buffered in the switch. This introduces a forwarding delay for each hop from source to destination. Due to the small synchronization error of the clocks between devices, the exact time when the entire frame has been received in a particular switch is unknown. Consequently, the time for forwarding the frame on the next link should take into account the worst-case synchronization error, δ.
- *Bounded end-to-end latency.* All TT flows must arrive within their relative deadline, i.e., the end-to-end latency cannot exceed the deadline. End-to-end latency is defined as the time from the sender initiates transmission of the first frame and until the last frame arrives at the receiver.

- *Deterministic queues.* Analogously to the *link congestion* property, a queue can be considered a shared resource, which can only be occupied by frames from a single flow at a time. In Fig. 4 this corresponds to the property that queue utilization boxes do not overlap in the time domain, if they belong to the same queue. In addition, there should be a δ-sized spacing between queue utilization boxes of frames arriving at different ingress ports, to account for the worst-case synchronization error.

Under these assumptions, the GCLs are conceptually equivalent to the schedule tables presented in Fig. 4. Fig. 4c and 4d shows two feasible schedules. GCLs can be optimized according to several criteria, e.g., TT queue usage and TT end-to-end latency. In order to improve queue usage, the frames of f_1 and f_2 should be rearranged in such a way that they both share the same queue in $[SW_1, ES_3]$ without occupying the queue at the same time. Fig. 4c shows such a schedule, where they both use q_1. Notice that the queue utilization boxes do not overlap.

On the other hand, minimizing queue usage has a negative effect on end-to-end latency. In Fig. 4c the frames of f_2 have been spaced further apart, thereby increasing the end-to-end latency. Instead, the schedule could be optimized with respect to end-to-end latency as shown in Fig. 4d. In this schedule, two queues are used but the frames of f_2 are grouped closer together compared to Fig. 4c resulting in a smaller end-to-end latency. This example shows that a desirable schedule is a tradeoff between queue usage and end-to-end latency.

Flows	Endpoints (v_s, v_t)	$P(B)$	$T(\mu s)$
f_1, f_2, f_3	$ES1 \Rightarrow ES4$	1500	100
f_4, f_5	$ES3 \Rightarrow ES2$	1500	100

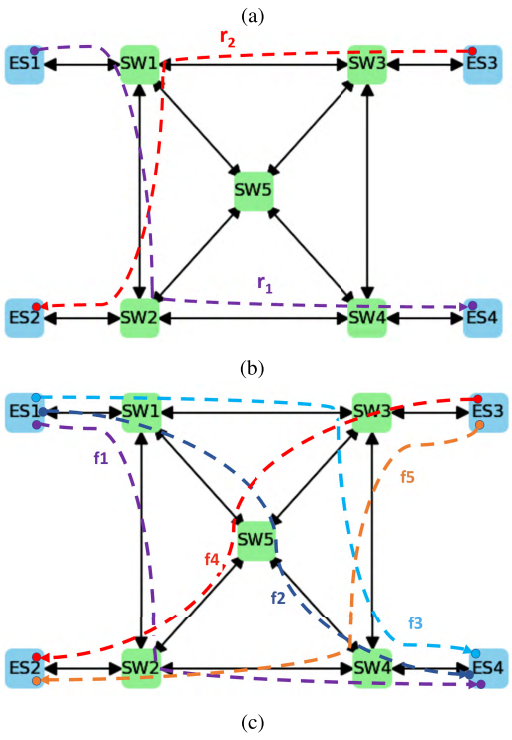


FIGURE 5. Example routing optimization for TT. (a) Flows example for routing. (b) Shortest path routing. (c) Optimized routing.

B. ROUTING FOR TT

Let us illustrate the importance of optimizing the routing for TT flows. Let us consider the TT flows in Fig. 5a to be routed and scheduled in the architecture depicted in Fig. 5, consisting of four end systems interconnected with five switches. If we use the shortest path routing, which would intuitively reduce the latency of TT flows, we obtain the two routes r_1 and r_2 as depicted in Fig. 5b. Flows f_1, f_2 and f_3 have the route r_1 and flows f_4, f_5 have the route r_2. In this situation, flow f_5 is not schedulable (does not meet its deadline), due to the congestion on the link $[SW_1, SW_2]$.

We can make the TT flows schedulable by rerouting one of the flows reducing thus the congestion on the link $[SW_1, SW_2]$. For example, flow f_1 can be routed on a longer route instead of the shortest path, e.g., $\{ES1, SW1, SW5, SW4, ES4\}$. The optimal routing, which minimizes the TT end-to-end latencies, is depicted in Fig. 5c (we label the routes with the flow numbers), which makes use of longer routes for several flows. This example shows that the routing of TT flows impacts their scheduling and has to be considered at the same time with the GCL synthesis.

C. AVB-AWARE TT ROUTING AND SCHEDULING

So far, we have ignored the AVB flows. Let us illustrate the importance of taking into account the AVB flows during the TT routing and scheduling. For this example we are using the network topology from Fig. 1a, which has three end systems, ES_1 to ES_3, and two switches, SW_1 and SW_2, implementing the set of flows presented in Fig. 1 with four TT and four AVB flows, respectively. All AVB flows are Class A, with the default idle slope of 75%. Each flow is packed in one frame and all links have a speed of 1 Gbps, resulting in a transmission time C of 6.33 μs for TT frames and 13.24 μs for AVB frames. In this example we consider the non-preemption integration mode.

A flow is schedulable if the frames arrive before their deadlines at the destination, even in the worst-case scenario captured by $WCD(f_i)$. For TT, the WCDs are determined directly by the GCLs. However, the WCD of an AVB flow f is determined by its *worst-case scenario*, i.e., the situation that delays f the most. An AVB frame will be delayed by other AVB and TT frames (including the guard band of those TT frames) sharing the same output port.

In the following examples we are going to show the schedule tables for TT flows and we will illustrate the worst-case scenario for one AVB flow, namely f_5.

See section III for how to read the schedule. In addition, the number on top of rectangle box represents the transmission time of the frame, and the number on top of a blank interval is the waiting time due to the negative credit of CBS or for timely block reasons. The Δ in the figure means

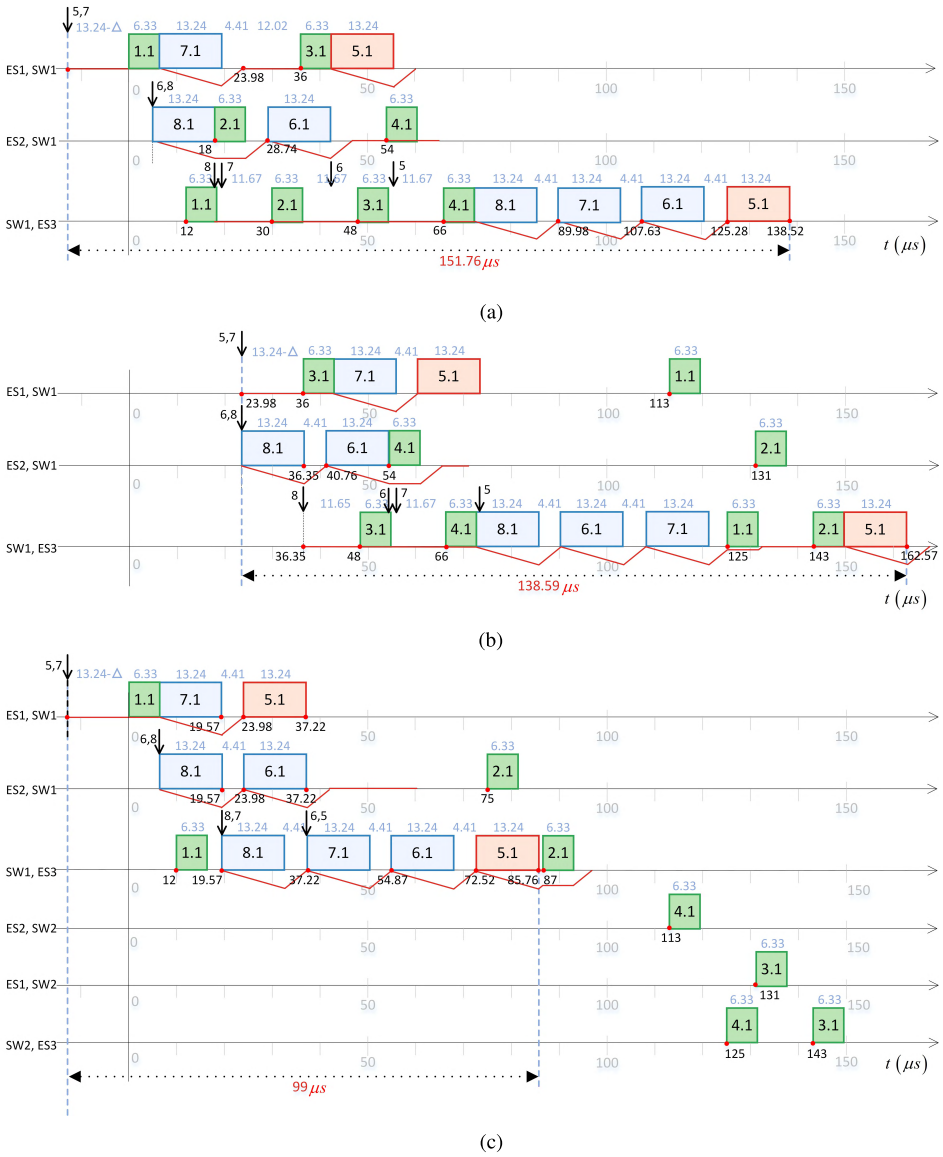


FIGURE 6. **Motivational examples for considering AVB during TT routing and scheduling. (a) Optimal TT schedule: AVB flows are unschedulable; WCD for f_5 is 151.76 μs. (b) AVB-aware TT schedule: WCDs of AVB flows are decreased to 138.59 μs. (c) Rerouting of some TT flows: *all* AVB flows are *schedulable*; WCDs for AVB flows are 99 μs.**

that the frame arrives just at the instant when the remaining time is slightly less than its transmission time. Similar to Fig. 3, we use downward pointing arrows to show the arrival times for AVB frames that create the worst-case scenario for f_5. Note that the GCLs are cyclic and in the worst case f_5 may arrive in the previous hyperperiod and it cannot start transmitting due to the timely block. The AVB credit for the queue of f_5 is depicted below the timelines using a red line.

Fig. 6a shows the optimal schedule for TT flows, which minimizes their WCDs by scheduling them as soon as possible. Only one queue is used for TT and all AVB flows are in the AVB Class A queue. As expected, the TT flows are schedulable, but all AVB flows have a WCD of 151.76 $\mu s > D$ so they are not schedulable. The worst-case scenario resulting in the WCD of 151.76 μs for f_5 is depicted in Fig. 6a.

However, if we construct an optimized GCL, as depicted in Fig. 6b, we can decrease the WCDs of all AVB flows to 138.59 μs. In this example, we have rescheduled the TT flows f_1 and f_2 by delaying them into the second half of the hyperperiod. This keeps the TT flows schedulable (still preserving their WCDs), but has the benefit of creating space for the AVB flows, decreasing their delay even in their worst-case. We illustrate the reduction of the WCD of AVB frame f_5 in Fig. 6b compared to Fig 6a.

Another choice is to reroute the TT flows f_3 and f_4 through switch SW_2 as depicted in Fig. 6c. As we can see, this preserves the schedulability of TT flows, as in the

previous examples, but now also all AVB flows are schedulable. By rerouting the TT traffic, the WCD of AVB flow f_5 becomes 99 $\mu s \leq D$.

The examples have shown the importance of carefully deciding the routes and GCLs for the TT traffic such that both TT and AVB are schedulable. Ignoring the AVB traffic when deciding on the routing and scheduling of TT, as is the case with all the related work, results in large AVB WCDs that miss the deadlines.

V. OPTIMIZATION STRATEGY

The problem presented in section IV is NP-hard. Exhaustively just enumerating every path between two vertices has been proven NP-hard [28]. Also, the corresponding decision problem of the TT scheduling problem, namely the flow-shop scheduling, is NP-complete [9]. Thus for a joint scheduling and routing problem to exhaustively evaluate every schedule and routing combination leads to an intractable amount of combinations that have to be evaluated. To solve our network design problem we use an integrated heuristic-metaheuristic strategy, i.e., a routing heuristic based on K-Shortest Path (KSP) method [29] and a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic [23] for the scheduling. The integrated strategy is further called Joint Routing and Scheduling *JRS* and is presented in Alg. 1.

Heuristics are not guaranteed to find the optimal solution; successful heuristics are able to find good quality solutions to large problem sizes in a reasonable time. Heuristics are algorithms specialized for a particular problem (in our case, routing), whereas metaheuristics are more general optimization algorithms that can be applied to a wide range of problems (we use the GRASP metaheuristic for scheduling).

JRS takes as an input the architecture G and the set of flows $\mathcal{F}$ and produces at the output a routing and scheduling solution ϕ. *JRS* generates multiple routing and scheduling alternatives, further called solutions (lines 3, 4) and evaluates each solution (line 5) attempting to find that solution which minimizes the *objective function*. The objective function is presented in section V-A. Our proposed routing and scheduling algorithms are presented in sections V-B and V-C, respectively. *JRS* terminates after a given time limit is reached or if there is no improvement in the objective function after a given number of iterations.

Algorithm 1 $JRS(G, \mathcal{F})$

1: $\phi \leftarrow \emptyset$
2: **repeat**
3: $R \leftarrow RoutingHeuristic(G, \mathcal{F})$
4: $\phi' \leftarrow SchedulingMetaheuristic(G, \mathcal{F}^{TT}, R)$
5: **if** $Obj(\phi', \mathcal{F}^{AVB}) < Obj(\phi, \mathcal{F}^{AVB})$ **then**
6: $\phi \leftarrow \phi'$
7: **end if**
8: **until** stopping criterion not met
9: **return** ϕ

A. OBJECTIVE FUNCTION

We are interested to find solutions that meet the deadlines for both TT and AVB flows. We consider a solution to be invalid if the TT deadlines are not satisfied. On each iteration of JRS and during the local search, the quality of a valid solution x is evaluated using an objective function $Obj(x, \mathcal{F}^{AVB})$ that checks if the AVB flows are schedulable, driving thus the search towards schedulable solutions:

$$Obj(x, \mathcal{F}^{AVB}) = \sum_{f_i \in \mathcal{F}^{avb}} \max(0, WCD(f_i) - f_i.D). \quad (1)$$

The objective function captures the sum of *tardiness* for each AVB flow, i.e., the time it takes to arrive at its destination after its deadline has passed.

To validate and evaluate a solution we need to compute the WCDs for both TT and AVB flows. Due to the deterministic behavior of TT traffic the WCD for a TT flow is easily computed from the schedule table as the difference between the time when the last frame arrives at the receiver and the time when the sender initiates transmission of the first frame. However, to determine the WCDs of AVB flows in presence of TT traffic, we have to use a schedulability analysis for AVB. Although such analyses have been proposed in the past, they have considered AVB in isolation, ignoring TT traffic. Only recently researchers have proposed AVB analysis to consider the influence of GCLs. Thus, reference [17] has extended the *AVB Latency Math* from *IEEE 802.1BA*, but this approach is very pessimistic. In this paper, we use the AVB schedulability analysis from [31] that: (i) is based on Network Calculus, (ii) takes into account the GCLs, (iii) supports both AVB Class A and B and (iv) considers all integration modes, considerably reducing the pessimism compared to [17].

B. ROUTING STRATEGY

Our routing heuristic, presented in Alg. 2, takes as input the architecture G and the set of TT flows $\mathcal{F}^{TT}$, including for each flow $f_i \in \mathcal{F}^{TT}$ the sending $f_i.v_s$ and receiving $f_i.v_t$ end systems. The strategy outputs the routing R which maps a route to each flow.

Initially the routing R is empty, which means that routes have to be found for all TT flows. As researchers demonstrate [17], [25], when we target time-sensitive applications, the shortest path routing may not lead to the smallest WCDs. However, enumerating all possible cycle-free paths (the full search space) to find the optimal routes is intractable. Hence, our strategy is to reduce the search space by using the K-Shortest Path (KSP) algorithm [29], which generates K unique routes of increasing length (the set R^{KSP}), starting from the shortest route. Our idea is that good quality routing solutions can be found by combining routes which, although are not the shortest routes, they are not excessively long. K is a parameter that controls how many routes are generated. The K parameter is randomly picked from the interval $[1, \overline{K}]$ (line 3), where $\overline{K}$ represents the upper bound of K, see [30] for details on how this upper bound can be determined experimentally for each flow based on the size of the topology.

Algorithm 2 *RoutingHeuristic*$(G, \mathcal{F})$

1: $R \leftarrow \emptyset$
2: **for** $f_i \in \mathcal{F}^{TT}$ **do**
3: $K \leftarrow PickK(f_i)$
4: $R^{KSP} \leftarrow GenerateRoutes(G, f_i.v_s, f_i.v_t, K)$
5: $R \leftarrow R \cup \{SelectRoute(R, f_i, R^{KSP})\}$
6: **end for**
7: **return** R

The routing heuristic iteratively selects a route for each TT flow attempting to evenly distribute the link utilization. The idea is that a high link utilization increases the WCDs. Thus, to determine a route $R(f_i)$ for a flow $f_i \in \mathcal{F}^{TT}$ the routing heuristic: (1) generates the reduced set of routes R^{KSP} line 4 and (2) selects the *least utilized* route from the generated set and adds it to the partial routing R (line 5). The utilization $U(R, R(f_i))$ of a route $R(f_i)$ from a partial routing solution R represents the maximum link utilization of the links composing the route, see section II-B for how we calculate the utilization.

If there are multiple routes with the minimum utilization, the heuristic selects a route randomly. By choosing randomly K and the least utilized route we are able to diversify the routing solutions output by our *RoutingHeuristic*, which helps *JRS* in Alg. 1 to explore the solution space.

C. SCHEDULING STRATEGY

GRASP is well-suited for combinatorial optimization problems, where an initial solution can be efficiently constructed in a greedy manner. Each iteration of GRASP consists of two phases: (1) A *construction phase*, where an initial feasible solution is built, and (2) a *search phase*, where a neighborhood around the initial solution is examined for improving solutions. The construction phase contributes with diversification, and the local search with intensification, enabling thus convergence towards a global optimum.

Alg. 3 presents our GRASP-based scheduling metaheuristic. As input, it takes (1) the architecture G, (2) the set of flows $\mathcal{F} = \mathcal{F}^{TT} \cup \mathcal{F}^{AVB}$, (3) the routing R and (4) two parameters, γ and π, related to the construction and local search phases, respectively. Alg. 3 outputs a feasible schedule x, which is the best scheduling solution found throughout the search in terms of the objective function from section V-A. Initially, x is empty (line 1), indicating that a feasible solution is yet to be found, i.e., the set of scheduled flows TT is empty.

Algorithm 3 *SchedulingMetaheuristic*$(G, \mathcal{F}, R, \gamma, \pi)$

1: $x \leftarrow \emptyset$
2: **repeat**
3: $x' \leftarrow GreedyRandomized(G, \mathcal{F}^{TT}, R, \gamma)$
4: $x' \leftarrow LocalSearch(x', G, \mathcal{F}^{TT}, R, \pi)$
5: **if** $Obj(x', \mathcal{F}^{AVB}) < Obj(x, \mathcal{F}^{AVB})$ **then**
6: $x \leftarrow x'$
7: **end if**
8: **until** termination criteria not met
9: **return** x

In each iteration, a new scheduling solution x' is generated in a greedy randomized fashion (line 3) using a constructive scheduling heuristic (*GreedyRandomized*, section V-C.1). *GreedyRandomized* contains a random element to ensure that different parts of the solutions space are explored in each iteration. The parameter γ defines the level of randomness. Too much randomness affects the quality of the initial schedules, whereas too little randomness affects diversification.

The initial solution is subsequently optimized via a local neighborhood search until reaching a local optimum (line 4). The local search destroys and repairs the current schedule to obtain new schedules. The parameter π specifies how much to destroy/repair in each iteration of the local search. If a new solution results in a better solution than the current best known, then the best solution is updated (lines 5 and 6). This repeats until a given execution time limit has been reached (termination criteria). The local search phase is described in section V-C3.

1) GREEDY RANDOMIZED HEURISTIC

The construction phase of the GRASP-based *SchedulingMetaheuristic* is presented in Alg. 4. *GreedyRandomized* is a polynomial-time greedy randomized heuristic designed to find feasible schedules which serve as good starting points for the subsequent local search. Hence, it should be computed efficiently, should not produce the same schedule in each iteration, and should not make obvious suboptimal decisions which the local search has to spend much time rectifying.

Flows are ordered by their period (line 2). To break ties, the route length is used as an indicator of how difficult flows are to schedule. Flows are scheduled one at a time in this order (lines 3-12). Given the current scheduling solution x, we attempt to schedule each unscheduled TT flow using several heuristic variations based on List Scheduling, see section V-C2. The Restricted Candidate List (RCL) (line 4) is a data structure that keeps track of the γ best schedules

Algorithm 4 *GreedyRandomized*$(G, \mathcal{F}^{TT}, R, \gamma)$

1: $x \leftarrow \emptyset$
2: $\mathcal{F}' \leftarrow SortByPeriod(\mathcal{F}^{TT}, R)$
3: **for** $f_i \in \mathcal{F}'$ **do**
4: $RCL \leftarrow RestrictedCandidateList(\gamma, f_i)$
5: **while** $ScheduleFlow(x, f_i, RCL) = true$ **do**
6: $x' \leftarrow x \cup \{f_i\}$
7: $RCL.AddCandidate(x')$
8: **end while**
9: **if** $RCL.Length() > 0$ **then**
10: $x \leftarrow RCL.GetRandomCandidate()$
11: **end if**
12: **end for**
13: **return** x

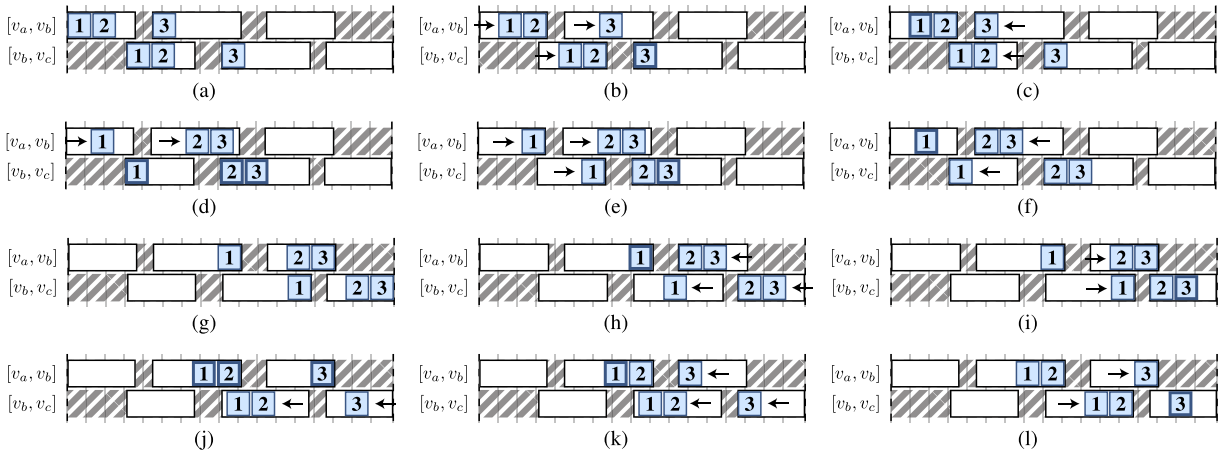


FIGURE 7. Heuristic variations for scheduling frames of a flow in an existing schedule. (a) ASAP. (b) ASAP-L. (c) ASAP-LF. (d) ASAPQ. (e) ASAPQ-L. (f) ASAPQ-LF. (g) ALAP. (h) ALAP-F. (i) ALAP-FL. (j) ALAPQ. (k) ALAPQ-F. (l) ALAPQ-FL.

produced by the heuristics with respect to the objective function *Obj*. When all heuristics have been considered, a random schedule is chosen from among those in RCL (line 10).

It may be the case that the heuristics are unable to schedule a particular flow. In that case, no candidate is added to RCL (lines 5-8). If all heuristics fail, RCL is empty (lines 9-11), i.e., that particular flow is not included in the schedule, making it invalid.

TT flows are scheduled individually using *ScheduleFlow*, presented in the next section.

2) SCHEDULE FLOW

Given an existing partial scheduling solution x, several feasible schedules exist for a flow f. In this section we present a List Scheduling-based heuristic approach called *ScheduleFlow* for scheduling the individual frames of a single flow, while minimizing queue usage. The achieved schedule can then, in turn, be post-processed to minimize end-to-end latency. The heuristic strategy is generalized into multiple variations denoted $\mathcal{H}$.

ScheduleFlow schedules the frames of f sequentially, scheduling each frame on all links before moving on to the next frame. It continues in this way until either all frames in f are successfully scheduled, or until failing to schedule a particular frame, i.e., failing to determine offsets for the frame such that all constraints of section IV-A are satisfied.

Fig. 7a illustrates an "As Soon As Possible" (ASAP) approach to scheduling frames in a schedule. A frame is scheduled on its route, in-order, at the earliest possible offset where the link is idle and the queue is empty. If the frame is not assigned to the same empty-queue block as it was on the previous link, the algorithm backtracks and reschedules the previous frame to the next empty-queue block. *ScheduleFlow* succeeds if the last frame is scheduled within the deadline and fails otherwise. Analogous to ASAP, an "As Late As Possible" (ALAP) approach can be formulated by traversing the frames in reverse order, as well as scheduling on links in reverse order. The reader is referred to [22], for more details about determining feasible frame offsets using the ASAP and ALAP approaches.

a: REDUCING QUEUE USAGE

To minimize queue usage, the heuristic initially assigns all flow instances to the first queue. If the heuristic at some point fails to schedule a particular frame on one of its links, it may be because the queue assignment imposes too many restrictions on the feasible frame offsets. In this case, the queue assignment is incremented for some link on the route, before restarting the algorithm from the first frame. The idle-link and empty-queue blocks are used to determine which link to increment. ASAP heuristic increments the first queue assignment which allows a frame to start earlier than with the current queue assignment. When the algorithm terminates one of two things has happened: Either all frames have been scheduled, or some switch has no more queues available, i.e., the heuristic failed to schedule the flow.

b: REDUCING END-TO-END LATENCY

Once a feasible solution has been found it can be post-processed to minimize end-to-end latency. Recall, that the end-to-end latency is the time from the offset of the first frame on the first link and until the finish time of the last frame on the last link. Hence, shifting the first frame to the right, or the last frame to the left reduces end-to-end latency. The intervals in which frames can safely be shifted without violating feasibility are computed from the empty-queue and idle-link intervals. A frame can be post-processed immediately when it has been scheduled on all links, or all frames can be post-processed together when the entire flow has been scheduled.

Fig. 7 shows heuristic variations including the original ASAP heuristic (Fig. 7a). White boxes represent the intervals where frames can be shifted. The main variation, ASAPQ, is illustrated in Fig. 7d. It reduces the time frames spend in

queues by shifting each frame instance as close as possible to the frame instance on the next link. Frame instances on the last link are not moved, which is illustrated with a thick border in Fig. 7d. As an important side effect, the method reduces the overall time the queue is occupied, which could lead to better queue utilization and a lower total number of queues.

The remaining ASAP variations are different ways of post-processing the offsets once all frames have been scheduled by either ASAP or ASAPQ. In ASAP-L (Fig. 7b) all frame instances are shifted toward the last frame instance to reduce end-to-end latency. The schedule produced by ASAP-LF (Fig. 7c) has been through an additional post-processing step, where all frames instances are shifted toward the first frame instance. ASAPQ-L and ASAPQ-LF are variations of ASAPQ that have been post-processed in the same two ways.

The same variations can be formulated for the ALAP heuristic, but every shift is reversed compared to ASAP. Consequently, the post-processing steps first move toward the *first* frame instance, then the *last*. Fig. 7g–7l depict the variations for ALAP. In total, twelve heuristic variants are used, targeting both the latency and the queue usage. We refer the reader to [22] for more details.

3) LOCAL SEARCH

The purpose of the local search phase is to intensify the search by investigating a well-defined neighborhood of solutions similar to the current solution. This corresponds to schedules where the majority of TT flows are scheduled exactly as in the current solution. It is likely that a better solution arises from rescheduling only a couple of TT flows. The local search attempts to identify such rearrangements by removing a small subset of TT flows, and rescheduling them in a different way.

The destroy and repair mechanisms of the local search rearrange flows compared to the original static order given by *SortByPeriod*. Thus, the local search can recover from a suboptimal ordering of flows in the construction phase.

The neighborhood is defined as follows: All the schedules which can be constructed by removing up to π flows from x, and subsequently rescheduling them using one of the scheduling heuristics. If a new, improving solution is discovered, the neighborhood search is repeated for the new solution. The local search continues in this way until reaching a local minimum, from which no solution from the neighborhood improves the current solution, or until exceeding the time limit.

VI. EXPERIMENTAL EVALUATION

We have performed three sets of experiments. In the first two sets of experiments we focus on the ability of our approach to determine good quality solutions for the TT routing and scheduling in a reasonable time. We are interested if our Joint Routing and Scheduling (JRS) approach scales well with large problem sizes. For these experiments we ignore AVB. Hence, in the first set of experiments (section VI-A) we evaluate the quality of our GRASP-based Scheduling Heuristic for TT, and in the second set of experiments we evaluate the importance of considering the TT routing during the GCL synthesis (section VI-B). In the last set of experiments we take AVB into account and evaluate JRS in terms of its ability to determine schedulable solutions for both TT and AVB.

TABLE 2. Comparison of ILP, OMT, and GRASP.

	running time (s)			*queue usage*			
ID	ILP	OMT	GRASP	k	$\underline{k}$	$\overline{k}$	k_N
T01	0.66	0.81	0.32	2	2	5	0
T04	2.49	2.46	0.21	2	2	5	0
T05	3.73	3.43	0.34	2	2	3	0
T10	4.70	5.12	0.72	4	4	8	0
T11	16.54	12.94	0.84	3	3	7	0
T12	210.03	34.33	0.69	5	5	9	0
T14	39.06	22.87	0.84	2	2	3	0
T18	10.98	7.17	0.56	2	2	5	0

A. EVALUATION OF GRASP-BASED SCHEDULING HEURISTIC FOR TT GCL SYNTHESIS

In the first set of experiments we were interested to evaluate the quality of our GRASP-based GCL synthesis approach. We have used the GRASP implementation from section V-C and we use two objective functions for GRASP, the normalized queue usage, denoted k_N, and the normalized end-to-end latency, denoted Λ_N. $\mathrm{k}_N(x)$ is a mapping of queue usage for TT traffic to the interval [0; 1] as shown in Eq. 2.

$$k_N(x) = \frac{k(x) - \underline{k}(x)}{\overline{k}(x) - \underline{k}(x)} \tag{2}$$

where $k(x)$ denotes the total number of queues used for TT traffic across all egress ports. $\underline{k}(\mathrm{x})$ and $\overline{k}(\mathrm{x})$ are lower and upper bounds, respectively. The lower bound is assuming a single TT queue in all egress ports forwarding TT traffic, and the upper bound assumes the minimum of the available queues in the egress port and the number of TT flows forwarded through that egress port.

The total end-to-end latency is normalized in Λ_N in a similar way. The lower bound assumes that all flows are scheduled independently, i.e., without interference from other TT flows. The upper bound assumes the worst-case scenario where all flows have end-to-end latencies equal to their deadline.

To evaluate GRASP, we have implemented the OMT approach from [5] and the ILP approach from [21], which both minimize the number of queues (k). The values of k are presented in the table, including the lower and upper bounds, and the normalized value. We have compared the three approaches on the test cases from [21], and our GRASP has been able to obtain the same optimal solutions in less than 1 second for all test cases as shown in table 2.

Further, we considered six topologies of varying size. The topologies are industrial sized, and are derived from the work presented in [19]. The topologies are grouped into three categories based on their size. There are three small topologies (G1, G2 and G3, with up to 4 ESes and 3 NSes), two medium (G4 and G5, with up to 48 ESes and 28 NSes), and one large (G6, with 256 ESes and 146 NSes), see Table 4 for the number

TABLE 3. Combinations of periods in test cases.

hyperperiod	short periods (μs)	long periods (ms)
1 ms	100, 200, 500	1
6 ms	100, 150, 200, 500	1, 2, 6
30 ms	100, 150, 200, 300, 500	5, 10, 30

TABLE 4. Average number of (flows, frames) of test cases.

	topology size		
hyperperiod	small	medium	large
1 ms	(17, 174)	(61, 548)	(358, 2078)
6 ms	(15, 436)	(55, 1193)	(254, 3682)
30 ms	(18, 737)	(63, 2944)	(327, 15167)

of flows on each topology size. The network precision is assumed to be $\delta = 5.008\ \mu s$. The transmission rate for all links is fixed at 1 Gbps, and the propagation delay of each link is assumed negligible, i.e., it is set to zero. Every egress port has eight queues.

The hyperperiod of all flows defines the width of the schedule, and has a major impact on the complexity of the problem. Thus, the hyperperiod is an important aspect to consider, when evaluating performance. We define three hyperperiods of 1 ms, 6 ms, and 30 ms. For each choice of hyperperiod we define a set of short periods and a set of long periods as presented in Table 3. Short-period flows have a data size of either one, two, or three times the MTU of 1500 bytes. Long-period flows have data sizes 10, 20, 40, 60, or 100 times MTU. The choice of periods and data sizes is inspired by [5].

In order to generate flows, that yield difficult scheduling problems in terms of queue usage and end-to-end latencies, the link utilization should be relatively high. Hence, synthetic applications are generated by repeatedly adding short-period and long-period flows to the set of flows. The sending and receiving end systems are randomly chosen among the end systems in the topology. This procedure is repeated until multi-queue scenarios arise.

For each choice of topology and hyperperiod, we generate 30 test cases with high link utilization. In total we use 540 test cases, 90 for each of the six topologies.

Table 4 shows the average number of flows and frames for every pair of topology class and hyperperiod. Overall, the test instances range from a few hundred frames to tens of thousands of frames.

We have extended the ILP formulation presented in [21], which minimizes queue usage, to also feature end-to-end latency minimization. For the ILP formulation, the Gurobi [1] solver was given a time limit of 4 hours, after which it returns the best feasible solution. The ILP approach is intractable for many of the test instances, especially for larger hyperperiods. The results are compared with GRASP in Fig. 8a and Fig. 8b for the subset of test cases solved by ILP (the x-axis shows the topologies G1 to G6, each with flows of varying hyperperiods, 1 to 30 *ms*). Some data points are missing, because the ILP approach was unable to find feasible solutions within the time limit.

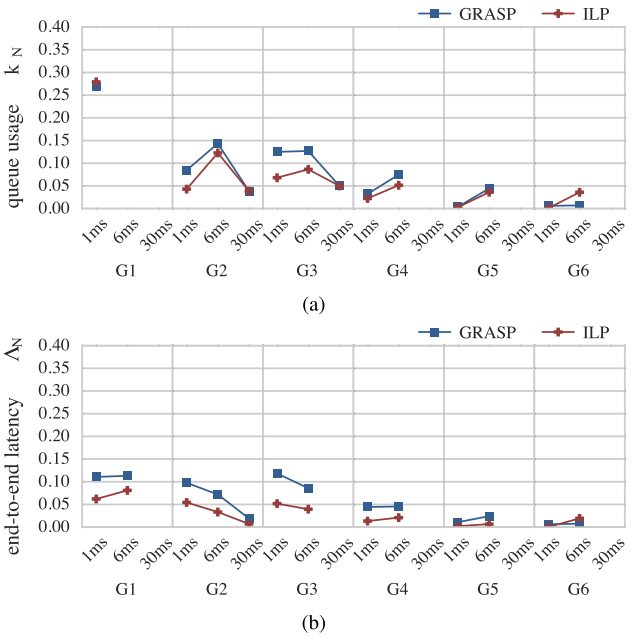


FIGURE 8. Comparison of GRASP and ILP. (a) Average queue usage for GRASP and ILP. (b) Average end-to-end latency for GRASP and ILP.

Fig. 8a shows on the y-axis the normalized queue usage k_N, and Fig. 8b shows in μs the normalized end-to-end latency Λ_N. The ILP approach was able to solve 48% of the instances when minimizing queue usage and 42% when minimizing latency. On average, the ILP approach produced schedules with 17% lower queue usage in Fig. 8a and 51% lower end-to-end latency in Fig. 8b, but had a 15–20 times longer execution time.

GRASP is able to significantly improve execution time compared to the ILP approach which is intractable for large instances, and is able to produce better schedules than a pure heuristic approach (e.g., variants of ASAP and ALAP). Its ability to minimize the objectives could be improved by increasing the time limit. Conversely, the time limit can be decreased in order to compute feasible schedules quickly. This flexibility makes GRASP well-suited to be used for GCL synthesis, where the schedules must take into account AVB flows. Our GRASP can also handle non-harmonic periods (that may result in large hyperperiods, which increase the complexity of the scheduling problem). However, in the case of extremely large hyperperiods we recommend adjusting the periods or changing the traffic type of the TT frames to AVB. In our previous work [11] we addressed the problem of determining the appropriate traffic type for each frame for mixed-criticality traffic.

B. EVALUATION OF TT ROUTING HEURISTIC

In this section we are interested to evaluate the ability of our *RoutingHeuristic* from Alg. 2 to find TT routes that improve the schedulability of TT frames. We have ignored AVB flows in these experiments and focused only on TT.

TABLE 5. Comparison of shortest path routing with our proposed TT *RoutingHeuristic*.

	TC1			TC2			TC3			TC4			Orion			SAE		
	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H
Shortest path	100	64	62	100	77	67	100	100	40	100	100	60	100	84	73	100	62	45
RoutingHeuristic	100	100	87	100	100	99	100	100	60	100	100	65	100	98	87	100	89	81

TABLE 6. Experimental results for *JRS*.

Test case	TT flows	AVB flows	End systems	Switches	TT latency		TT Queues		TT+AVB		*JRS*	
					Exec.	Sched.	Exec.	Sched.	Exec.	Sched.	Exec.	Sched.
Motiv.	4	4	3	2	1.1	No	0.8	No	32	Yes	21.9	Yes
TC5	12	4	5	2	2.2	No	2.6	No	403.9	Yes	148.3	Yes
TC6	35	26	32	18	8.7	No	9.6	No	612.8	Yes	527.7	Yes
TC7	427	464	256	146	708.6	No	628.8	No	728.5	No	534.6	Yes
Auto.	14	34	20	20	2.8	No	2.7	No	422.8	No	300.03	Yes

We have used four synthetic test cases, TC1–TC4 and two real-life test cases, "Orion" and "SAE". We have varied the size and type of network topology, i.e., topologies such as "mesh" that have more alternative routes connecting ESes and topologies with less alternative routes. Thus, TC1 is a mesh topology with 6 end systems and 8 network switches and TC2 is a mesh topology with 12 ESes and 14 NSes. TC3 and TC4 use a "ring" topology that has fewer alternative routes. In TC3, we have added more dataflow links such that TC3 is in-between a mesh and a ring topology. Both TC3 and TC4 have 12 ESes and 12 NSes. Finally, "Orion" and "SAE" are real-life topologies adapted from [26].

The results are presented in table 5, where we have used the results obtained considering "Shortest path" routes as a baseline. In the table we show three setups for each test case: low utilization (L), medium utilization (M) and high utilization (H). The increasingly higher utilization has been obtained by increasing the number of TT flows; the TT flows were generated as presented in section VI-A. For each algorithm and test case, we show the percentage of TT flows that are schedulable (out of 100%).

As we can see from table 5, as the size of the topology and the utilization increase, the shortest path routing has difficulties in finding schedulable solutions for the TT flows. Note that, for low utilizations it is easy to find schedulable solutions even without optimizing the routing. However, our *RoutingHeuristic* is able to significantly improve on the shortest path routing as the topologies get larger and more utilized.

As we can see from the results, for topologies that have alternative routes, optimizing routing is very important if we want to obtain solutions where TT flows can be scheduled, and our *RoutingHeuristic* is able to find such results in the short time limits imposed. We have used time limits of 1, 5, 15 and 30 minutes, corresponding to the topology size and utilization (the larger the topology and utilization, the longer the time limit). As expected, for topologies that do not have multiple alternative routes between end systems (e.g., TC3 and TC4), optimizing routing is less important.

C. EVALUATION OF AVB-AWARE ROUTING AND SCHEDULING

Our proposed TT routing and scheduling algorithms are the only approaches proposed so far in the literature that can take into account the AVB flows. Our AVB-aware Joint Routing and Scheduling (*JRS*) for TT flows is evaluated in this section. We have considered the non-preemption integration mode in these experiments, but our approach is able to handle all integration modes.

To evaluate *JRS*, we used five test cases, "Motiv." (the motivational example in Fig. 6), three synthetic test cases, TC5–7 and an automotive test case "Auto.". For TC5–7 we used star and snowflake topologies, gradually increasing the size of the network. For "Auto", we used traffic flows obtained from an automotive manufacturer, implementing ADAS functions, and we have used the approach from [10] to generate the network topology. The details of the test cases, i.e., the number of TT and AVB flows and the number of ESes and NSes are in table 6 columns 2–5.

We have run our proposed *JRS* optimization strategy on these test cases, and the results are presented in table 6 under the heading *JRS*. We have compared our approach with three other approaches. (1) Thus, "TT+AVB" does not attempt to optimize routing (considers shortest paths) and instead uses the *SchedulingMetaheuristic* from Alg. 3 to determine GCLs such that both TT and AVB flows are schedulable (i.e., it takes into account the AVB flows).

The two other approaches ignore AVB flows and also use shortest path routing. Thus, (2) in the "TT latency" approach, where we minimized the latency of TT flows and (3) in the "TT queues" approach we minimized the number of TT queues (in the hope of helping indirectly the AVB flows by reducing the number of higher-priority TT queues). These two approaches were also implemented using *SchedulingMetaheuristic*, but the objective function has been changed to minimize the TT latency and the number of TT queues, respectively, see section VI-A. The "Exec." columns show the algorithms' runtime in seconds. The TT flows were schedulable for all experiments. In the columns labeled "Sched." we show if the AVB flows were also schedulable.

As we can see from the table, if AVB is ignored during scheduling as is the case with the ''TT latency'' and ''TT queues'' approaches, we are not able to find schedulable solutions for AVB flows. However, if we take into account the AVB flows during the scheduling (the ''TT+AVB'' heading in table 6), we are able to find solutions where both TT and AVB flows are schedulable. However, this is possible only for the smaller test cases, ''Motiv.'', TC5 and TC6. For the larger test case TC7 and the realistic test case ''Auto.'', we also have to optimize the routing of the TT flows in order to obtain schedulable solutions.

The conclusion is that only by using our *JRS* approach that takes the AVB into account during the routing and scheduling optimization, we are able to obtain schedulable solutions where all AVB flows are also schedulable. In addition, our approach is able to handle large problem sizes, such as TC7, with a network of 402 devices and 891 flows.

VII. CONCLUSIONS

In this paper we have considered TSN-based cyber-physical systems and running mixed-criticality real-time applications that use both TT and AVB traffic. We were interested to decide the routing, the number of TT queues, the allocation of TT flows to TT queues and the GCLs of TT queues such that all flows are schedulable. Our approach was to use an integrated heuristic-metaheuristic, called Joint Routing and Scheduling (JRS), to search for TT routing and scheduling solutions where both TT and AVB flows are schedulable. Each solution visited during the search was evaluated in terms of AVB schedulability using a Network Calculus approach that takes into account the impact of TT GCLs on the WCDs of AVB flows. Our results show that only by taking into account the AVB flows during the routing and GCL synthesis we can obtain schedulable solutions for both TT and AVB.

ACKNOWLEDGEMENTS

Dr. Silviu S. Craciunas, TTTech Computertechnik AG, has contributed to the TT scheduling problem formulation, the synthesis of test cases and the design of the experimental evaluation for the GRASP-based scheduling approach. This has enabled us to perform the comparison to the optimal results obtained by the OMT approach from his related work. He has also provided feedback on the related text. Cristian Zara, master's student at DTU, has contributed the routing code and the evaluation of TT routing impact on scheduling.

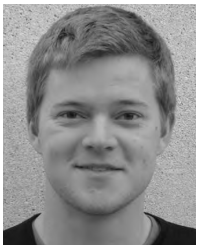
REFERENCES

[1] (Oct. 2017). *Gurobi Optimizer*. [Online]. Available: http://www.gurobi.com/products/gurobi-optimizer
[2] Aeronautical Radio Inc., ''ARINC specification 664P7, aircraft data network, Part 7, avionics full duplex switched Ethernet (AFDX) network,'' Aeronautical Radio Inc., Annapolis, MD, USA, Tech. Rep., 2005.
[3] Aeronautical Radio Inc., ''Aircraft data network, part 7, avionics full-duplex switched Ethernet network,'' ARINC, Annapolis, MD, USA, Tech. Rep., 2009.
[4] S. S. Craciunas and R. S. Oliver, ''Combined task- and network-level scheduling for distributed time-triggered systems,'' *Real-Time Syst.*, vol. 52, no. 2, pp. 161–200, 2016.
[5] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, ''Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks,'' in *Proc. Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2016, pp. 183–192.
[6] J.-D. Decotignie, ''Ethernet-based real-time and industrial communications,'' *Proc. IEEE*, vol. 93, no. 6, pp. 1102–1117, Jun. 2005.
[7] F. Dürr and N. G. Nayak, ''No-wait packet scheduling for IEEE time-sensitive networks (TSN),'' in *Proc. Int. Conf. Real-Time Netw. Syst. (RTNS)*, Oct. 2016, pp. 203–212.
[8] *FlexRay Communications System Protocol Specification Version 3.0.1*, FlexRay Consortium, 2010.
[9] M. R. Garey, D. S. Johnson, and R. Sethi, ''The complexity of flowshop and jobshop scheduling,'' *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, 1976.
[10] V. Gavriluţ, B. Zarrin, P. Pop, and S. Samii, ''Fault-tolerant topology and routing synthesis for ieee time-sensitive networking,'' in *Proc. RTNS*, Oct. 2017, pp. 267–276.
[11] V. Gavrilut and P. Pop, ''Traffic class assignment for mixed-criticality frames in TTEthernet,'' *ACM Sigbed Rev.*, vol. 13, no. 4, pp. 31–36, 2016.
[12] V. Gavrilut and P. Pop, ''Scheduling in time sensitive networks (TSN) for mixed-criticality industrial applications,'' in *Proc. Int. Workshop Factory Commun. Syst. (WFCS)*, 2018, pp. 1–4.
[13] (2011). *IEEE 802.1BA—Audio Video Bridging (AVB) Systems, Draft 2.5*. [Online]. Available: http://www.ieee802.org/1/pages/802.1ba.html
[14] (2015). *IEEE 802.3 Standard for Ethernet*. [Online]. Available: http://www.ieee802.org/3/
[15] (2016). *IEEE Official Website of the 802.1 Time-Sensitive Networking Task Group*. [Online]. Available: http://www.ieee802.org/1/pages/tsn.html
[16] D. Jansen and H. Buttner, ''Real-time Ethernet: The EtherCAT solution,'' *Comput. Control Eng.*, vol. 15, no. 1, pp. 16–21, 2004.
[17] S. M. Laursen, P. Pop, and W. Steiner, ''Routing optimization of AVB streams in TSN networks,'' *ACM Sigbed Rev.*, vol. 13, no. 4, pp. 43–48, 2016.
[18] N. G. Nayak, F. Dürr, and K. Rothermel, ''Routing algorithms for IEEE802. 1Qbv networks,'' *ACM Sigbed Rev.*, vol. 15, no. 3, pp. 13–18, 2017.
[19] R. S. Oliver, S. S. Craciunas, and G. Stöger, ''Analysis of deterministic Ethernet scheduling for the industrial Internet of Things,'' in *Proc. IEEE 19th Int. Workshop Comput. Aided Modeling Design Commun. Links Netw.*, Dec. 2014, pp. 320–324.
[20] M. Pahlevan, N. Tabassam, and R. Obermaisser, ''Heuristic list scheduler for time triggered traffic in time sensitive networks,'' *ACM Sigbed Rev.*, to be published.
[21] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, ''Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks,'' *IET Cyber-Phys. Syst., Theory Appl.*, vol. 1, no. 1, pp. 86–94, 2016.
[22] M. L. Raagaard and P. Pop, ''Optimization algorithms for the scheduling of IEEE 802.1 time-sensitive networking (TSN),'' Tech. Univ. Denmark, Lyngby, Denmark, Tech. Rep., Jan. 2017.
[23] M. G. Resende and C. C. Ribeiro, ''GRASP: Greedy randomized adaptive search procedures,'' in *Search Methodologies*. Boston, MA, USA: Springer, 2014, pp. 287–312.
[24] SAE International, ''Time-triggered Ethernet,'' SAE Tech. Paper AS6802, 2011.
[25] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, ''ILP-based joint routing and scheduling for time-triggered networks,'' in *Proc. Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2017, pp. 8–17.
[26] D. Tămaş-Selicean, P. Pop, and W. Steiner, ''Design optimization of TTEthernet-based distributed real-time systems,'' *Real-Time Syst.*, vol. 51, no. 1, pp. 1–35, 2015.
[27] D. Thiele and R. Ernst, ''Formal worst-case performance analysis of time-sensitive Ethernet with frame preemption,'' in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2016, pp. 1–9.
[28] L. G. Valiant, ''The complexity of enumeration and reliability problems,'' *SIAM J. Comput.*, vol. 8, no. 3, pp. 410–421, 1979.
[29] J. Y. Yen, ''Finding the K shortest loopless paths in a network,'' *Manage. Sci.*, vol. 17, no. 11, pp. 712–716, 1971.
[30] C. Zara, ''Routing algorithms for real-time traffic on IEEE 802.1 time-sensitive networking,'' M.S. thesis, DTU Compute, Tech. Univ. Denmark, Lyngby, Denmark, 2018.

[31] L. Zhao, P. Pop, Z. Zheng, and Q. Li, "Timing analysis of AVB traffic in TSN networks using network calculus," in *Proc. Real-Time Embedded Technol. Appl. Symp. (RTAS)*, 2018, pp. 25–36.

VOICA GAVRILUŢ received the M.Sc. degree in distributed systems in Internet from the Babeş-Bolyai University, Cluj-Napoca, Romania, in 2014. She is currently pursuing the Ph.D. degree with the Technical University of Denmark. Her interests are on network design methods for mixed-criticality and safety-critical applications.

LUXI ZHAO received the Ph.D. degree in communication and information system from Beihang University, Beijing, China, in 2017. She has been holding a post-doctoral position with the Technical University of Denmark since 2017. Her main research interest concerns worst-case analysis and performance evaluation on real-time and safety-critical networks.



MICHAEL L. RAAGAARD received the M.Sc. degree in computer science and engineering from the Technical University of Denmark in 2017. His main research interest is combinatorial optimization in embedded systems design. He has applied his research to the areas of microfluidic biochip synthesis and safety-critical networks configuration.

PAUL POP received the Ph.D. degree in computer systems from Linköping University in 2003. He has been an Associate Professor at DTU Compute, Technical University of Denmark, and since 2016, he has been a Professor of cyber-physical systems. His research is focused on developing methods and tools for the analysis and optimization of dependable embedded systems. In this area, he has published over 130 peer-reviewed papers, three books and seven book chapters. His research has been highlighted as "The Most Influential Papers of 10 Years DATE". He has served as technical program committee member on several conferences, such as DATE and ESWEEK. He has received the best paper award at DATE 2005, RTIS 2007, CASES 2009, MECO 2013, and DSD 2016. He has also received the EDAA Outstanding Dissertations Award (co-supervisor) in 2011. He is the Chairman of the IEEE Danish Chapter on Embedded Systems. He is the Director of the DTU's IoT Research Center and has coordinated the Danish national InfinIT Safety-Critical Systems Interest Group.